On Efficient and Effective Association Rule Mining from XML Data

Ji Zhang[1], Tok Wang Ling[2] , Robert M. Bruckner[3], A Min Tjoa[4], Han Liu[1]

[1]*Department of Computer Science, University of Toronto,*
Toronto, Ontario, M5S 3G4, Canada
Email: {jzhang, hanliu}@cs.toronto.edu
[2]*Department of Computer Science, National University of Singapore, Singapore 117543*
Email: lingtw@comp.nus.edu.sg
[3]*Microsoft Research, Redmond, WA, USA*
Email: robruc@microsoft.com
[4]*Institute of Software Technology, Vienna University of Technology, Vienna, Austria*
Email: tjoa@ifs.tuwien.ac

Abstract: In this paper, we propose a framework, called XAR-Miner, for mining ARs from XML documents efficiently and effectively. In XAR-Miner, raw XML data are first transformed to either an Indexed Content Tree (IX-tree) or Multi-relational databases (Multi-DB), depending on the size of XML document and memory constraint of the system, for efficient data selection in the AR mining. Concepts that are relevant to the AR mining task are generalized to produce generalized meta-patterns. A suitable metric is devised for measuring the degree of concept generalization in order to prevent under-generalization or over-generalization. Resultant generalized meta-patterns are used to generate large ARs that meet the support and confidence levels. An efficient AR mining algorithm is also presented based on candidate AR generation in the hierarchy of generalized meta-patterns. The experiments show that XAR-Miner is more efficient in performing a large number of AR mining tasks from XML documents than the state-of-the-art method of repetitively scanning through XML documents in order to perform each of the mining tasks.

1. Introduction

The fast-growing amount of XML-based information on the web has made it desirable to develop new techniques to discover patterns and knowledge from the XML data. Association Rule (AR) mining is frequently used in data mining to reveal interesting trends, patterns, and rules in large datasets. Mining ARs from XML documents thus has become a new and interesting research topic. Association rules have been first introduced in the context of retail transaction databases [1]. Formally, an AR is an implication in the form of $X \rightarrow Y$, where X and Y are sets of items in database D that satisfy $X \cap Y = \Phi$, where D is a set of data cases. X and Y are called the antecedent and subsequent of the rule, respectively. The rule $X \rightarrow Y$ has support of s in D if $s\%$ of the data cases in D contain both X and Y, and the rule has a confidence of c in D if $c\%$ of the data cases contain Y if they also contain X. Association rule mining aims to discover all large rules whose support and confidence exceed user-specified minimum support and confidence thresholds: *minsup* and *minconf*.

A recent trend of mining ARs is the development of query language to facilitate association rule mining. The MINE RULE operator is introduced in [8] and extended in [9]. In addition, [7] introduced a SQL extension, called MSQL, which is equipped with various operators over association rules such as generation, selection, and pruning. DMSQL [5] is another query language used for association rule specification. There has

already been research in mining ARs from semi-structured documents [2, 4, 12, 13], among which the works in [2, 4] use unsupervised procedures and [12, 13] use single and multi-constraints in mining ARs. The constraint-based methods, compared to the non-constraint ones, are able to achieve better efficiency in the mining process and generate a manageable number of rules. In the domain of XML data, the work in [3] uses the MINE RULE operator introduced in [8] for AR mining purposes in native XML documents. The Predicative Model Markup Language (PMML) is proposed to present various patterns such as association rules and decision trees extracted from XML documents [10]. [15] presents a XML-enable AR mining framework, but does not give any details on how to implement this framework efficiently. [16] claims that XML AR can be simply accomplished using XQuery language.

However, the major problems with the state-of-the-art methods, i.e. [3], [10] and [16], are two-fold: (i) they select data from native XML data, thus the efficiency of these approaches is low because of the normally huge volume of XML data need to scanned in the process of AR mining. This becomes even worse when performing a large number of AR mining tasks. Data organization in XML format inherently renders the data selection inefficient. (ii) They do utilize generalization (e.g. group or cluster operations) of data. However, they lack the mechanism to control the degree to which generalization is performed. Under-generalization or over-generalization will seriously affect the effectiveness of the AR mining.

To address the above problems, we will propose a framework, called XAR-Miner, to efficiently and effectively mine ARs from XML documents in this paper. In this framework, XML data are extracted and organized in a way that is suitable for efficient data selection in the AR mining. Concepts relevant to the AR mining task are generalized, if necessary, to a proper degree in order to generate meaningful yet nontrivial ARs.

2. Overview of the Framework

The framework of AR mining of XML data consists of three major parts: (1) Pre-processing (i.e., construction of the Indexed XML Tree (IX-tree) or Multiple Relational Databases (Multi-DB)); (2) Generation of generalized meta-patterns; and (3) Generation of large ARs of generalized meta-patterns.

In the pre-processing module, data in the XML document are extracted and the IX-tree or Multi-DB will be built. Choice of IX-tree or Multi-DB is made based on the size of XML data and the main memory constraint of the system. IX-tree will be used when the XML data can be fully loaded into the main memory of the system while Multi-DB will be used otherwise. The transformed XML data will be indexed using sorting method to facilitate efficient data selections during the mining process.

Secondly, the AR mining task issued by user is analyzed and properly formulated by the system. Because of the sparsity of the XML data, especially in the AR mining tasks that involve multiple concepts, concept generalization will usually be performed to generate generalized meta-patterns. Two constraints, called *Min_gen* and *Max_gen*, are used to specify the allowed degree of generalization, which help avoid under-generalization or over-generalization of the meta-patterns.

Large ARs of the generalized meta-patterns that meet the support and confidence requirements are then mined using Apriori algorithm, a typical AR mining algorithm, in the AR Mining Module.

In this paper, a sample XML document of employee information, *EmployeeInfo.xml* (shown in Figure 1), is used to illustrate the ideas, definitions and algorithms of XAR-Miner, as appropriate.

```
<EmployeeInfo>
        <Employee>
                <Name> James Wang </Name>
                <Education>
                        <Major> Computer Science </major>   <Degree> B.S </Degree>
                        <Major> Computer Science </major>   <Degree> M.Sc </Degree>
                </Education>
                <TypeOfJob> System Analyst </TypeOfJob>
        </Employee>
        <Employee>
                 <Name> Ghai Vandana </Name>
                <Education>
                        <Major> Accounting </major>     <Degree> B.A </Degree>
                 </Education>
                <TypeOfJob> Accountant </TypeOfJob>
        </Employee>
        <Employee>
                  <Name> Linda Lee </Name>
                  <Education>
                        <Major> Mathematics </major>   <Degree> B.S </Degree>
                        <Major> Management </major>   <Degree> M.B.A </Degree>
                        <Major> Economics </major>      <Degree> Ph.D </Degree>
                  </Education>
                <TypeOfJob> Project Manager </TypeOfJob>
         </Employee>
</EmployeeInfo>
```

Fig. 1. Sample XML document (EmployeeInfo.xml)

3 XML Data Extraction and Transformation

The preprocessing work of XAR-Miner is to extract information from the original XML document and transform them into a way that is suitable for efficient AR mining. Specifically, we build *Indexed XML Tree* (IX-tree) when all the XML data can be loaded into main memory, and build *Multi-relational Databases* (Multi-DB) otherwise. We will evaluate the size of the XML data involved and the main memory available to select the proper strategy for XML data transformation and storage before AR mining tasks are preformed.

3.1 Construction of Indexed XML Tree (IX-tree)

Definition 1. *Indexed XML Tree (IX-tree)*: An Indexed XML Tree (IX-tree in short) is a rooted tree IX-tree=<*V, E, A*>, where *V* is the vertex set, *E* is the edge set and *A* is the indexed array set. *V* is set of nodes appearing in the XML document. The intermediate nodes in the IX-tree store the addresses of its immediate parent and children. An edge $e(v_1,v_2)$ in the IX-tree connects the two vertices v_1 and v_2 using a bi-directional link. The set of indexed arrays *A* positioned at the bottom level in the IX-tree stores the data in the leaf element or attribute nodes in the original XML document.

The IX-tree of EmployeeInfo.xml is shown in Figure 2. The IX-tree can be seen as a sort of hybrid of hierarchical and array-like structure. The hierarchical structure maintains the inherent parent-child and sibling relationships of the extracted XML data while the arrays store their indexed structural values. The construction of the IX-tree consists of the following two steps:

(1) Hierarchical structure building of IX-tree

The XML document will be scanned to get hierarchical information and structural values in the leaf nodes of the document. The moment an element or attribute is scanned in the XML document, a new node will be created in the corresponding IX-tree. Two pointers, *ParentNode* and *ChildNode* are used in this scanning process. If *ParentNode* is currently not a leaf node, the next node (its child node) will be read and these two nodes will be connected using a bi-directional link in the IX-tree. When the leaf node has been reached, the data in this leaf node will be retrieved and stored in an array. This process will be terminated when the end of the XML document is reached.

(2) Indexing of data in the arrays

The arrays storing the extracted data from the XML document are indexed by sorting the data to facilitate fast selection of relevant data for the AR mining task.

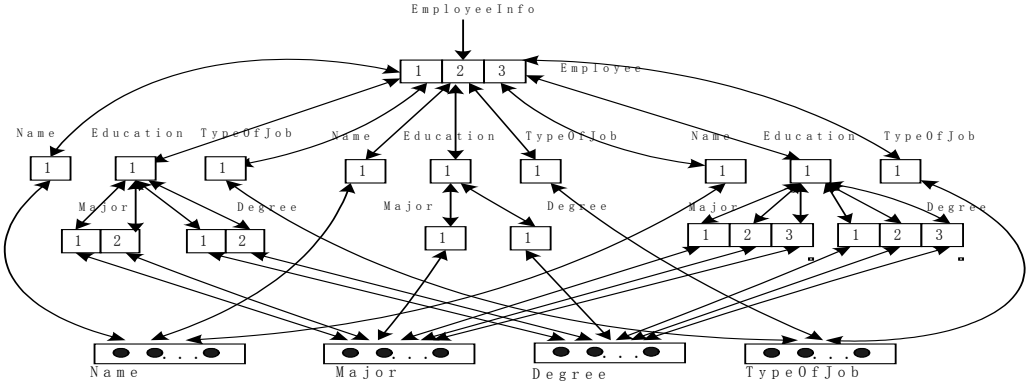

Fig. 2. IX-tree of EmployeeInfo.xml

3.2 Construction of Multi-relational Databases of XML Data

Under the circumstance that the size of XML data exceeds the main memory available, it is impossible to build an in-memory IX-tree. Therefore, we alternatively choose to construct Multi-relational Databases to accommodate the extracted XML data. In this architecture, an XML document will be transformed to a few relational databases, each of which will store the indexed structural values of a concept in the XML document. Unlike using hierarchical structure to maintain the inherent parent-child and sibling relationships of XML data, we use the notion of Serial Xpath String (SXS) to identify each XML data in relational databases.

Definition 2. *Serial Xpath String (SXS) of XML data*: The Serial Xpath String of an XML data x, denoted as SXS(x), is a string that gives the ordinal numbers of concepts of different levels along the path from the root to the leaf node of x. Any SXS will start with the root node and the ordinal number of concepts along the path will be delimited by dashed lines.

For example, the SXS of the data "Economics" in Figure 1 is "R-3-1-3". R in the SXS denotes the root node of the hierarchy. 3, 1 and 3 are the ordinal number of the concepts along the path, i.e. Employee, Education and Major. Please note an SXS with in a relational database is unique but not so across different databases since an SXS itself does not hold any information regarding the concepts along the path. To uniquely identify XML data stored in the multi-relational databases, we use the complete path in the XML tree starting from the root to the leaf node of the XML data, called *Xpath*, and associate it with each of the relational databases. In this particular example, the complete

path of "Economics' which is associated with Major.db is Employeeinfo/Employee/Education/Major, thus the SXS of "R-3-1-3" is equivalent to Employinfo-Employee(3)-Education(1)-Major(3), meaning that Economics is the third major (i.e. PhD) in the education information of the third employee in the XML document. The salient feature of SXS is that it is compact and, together with the Xpath associated with each relational database, it is able to uniquely identify an XML data in the whole XML document using the ordinal occurrence number of concepts in SXS. In addition, by using a string format, it is able to well maintain the hierarchical relationships of XML data. The Multi-relational databases transformation of the Employeeinfo.xml is given in Figure 3.

Name.db (XPath: Employeeinfo/Employee/Name)

Name	SXS
James Wang	R-1-1
Ghai Vandana	R-2-1
Linda Lee	R-3-1

TypeOfJob.db (XPath: Employeeinfo/Employee/TypeOfJob)

TypeOfJob	SXS
System Analyst	R-1-1
Accountant	R-2-1
Project Manager	R-3-1

Major.db(XPath: Employeeinfo/Employee/Education/Major)

Major	SXS
Computer Science	R-1-1-1
Computer Science	R-1-1-2
Accounting	R-2-1-1
Mathematics	R-3-1-1
Management	R-3-1-2
Economics	R-3-1-3

Degree.db (XPath: Employeeinfo/Employee/Education/Degree)

Degree	SXS
B.S	R-1-1-1
M.Sc	R-1-1-2
B.A	R-2-1-1
B.S	R-3-1-1
M.B.A	R-3-1-2
Ph.D	R-3-1-3

Fig. 3. Multi-relational database transformation of Employeeinfo.xml

3.3 Advantages of XML Data Transformation

The major advantage of performing XML Transformation into either IX-tree or Multi-DB architecture is that this transformation can ideally satisfy the requirements of AR mining as follows: firstly, AR mining often involves selecting data that are only relevant to the AR mining task. These data are usually only a fragment of the whole XML document. Information indexing in each of the arrays or relational databases helps to greatly facilitate the retrieval of relevant data from the relational databases for AR mining. The selection of data from these indexed structures is much more efficient than selecting data from the original XML document that usually requires a scan through the whole XML document. Secondly, the relationships among data in the XML document, such as a parent-child relationship or a sibling relationship, are perfectly maintained in the transformation by using either hierarchical structure in IX-tree or SXSs in each of the relational databases, enabling us to meet any AR mining requirement on the inherent hierarchical structure of the XML document.

4 Generate Large AR Rules

4.1 Data Selection from IX-tree and Multi-DB

One of key problems we have to deal with in the AR mining from XML document is the selection of data that are relevant to the AR mining task. In this step, we will select all the instances of involved concepts that meet the WHERE specification. This step

resembles the answering of queries. Since all the data in IX-tree and the Multi-relational databases have been indexed, the retrieval of qualified data from them can be very efficient. This efficiency is very significant when the number of AR mining tasks to be performed is large. If the WHERE statement involve multiple concepts, then the query results of multiple database will be joined to generate the final result of this step.

(a) Data Selection in IX-tree

Recall that in IX-tree, we take advantage of the bi-directional linking between parent and child nodes in the tree hierarchy to realize fast top-down and bottom-up traversal. This fast traversal can facilitate the retrieval of the value of the concepts involved in AR mining. We classify the concepts in the AR mining as *queried concepts* if they are used to specify the scope of data in the WHERE clause and *non-queried concepts* otherwise. We will use the path connecting the queried concepts used in the WHERE clause for data retrieval of non-queried concepts. To create such a path, the Nearest Common Ancestor Node (NCAN) of these concepts must first be found.

Definition 2. ***Nearest Common Ancestor Node (NCAN)***: The NCAN of elements $e_1,e_2,...,e_n$ in the hierarchy of an XML document H, denoted as $\text{NCAN}_H(e_1,e_2,...,e_n)$, is defined as the common ancestor element node in H that is the closest to e_i ($1 \leq i \leq$ n).

The NCAN in the hierarchy serves as the common information bridge semantically linking together data of the elements involved in an AR mining task. It is also called the *most informative subsumer* [11] of nodes in the tree structure. For example, the NCANs of the various nodes in the hierarchy in Figure 2 are as follows: $NCAN_H$(Education, TypeOfJob)=Employee and $NCAN_H$(Major, Degree)=Education.

Searching a NCAN of concepts in a XML hierarchy involves path comparison for these concepts. The searching of NCAN can be divided into two sub-steps. The first step is to find the common nodes of all the paths and the second step is to find the node from these common nodes that is closet to the leaf nodes of all these paths.

After the NCAN has been determined, instances of non-queried concepts will be retrieved by first performing a bottom-up traversing from the queried concepts to their NCAN and then a top-down traversing from their NCAN to the non-queried concepts.

(b) Data Selection in Multi-DB

Unlike IX-tree, Multi-DB architecture does not have the bi-directional link among nodes for traversal. However, we have created SXS for each XML data and Xpath for each relational database that perfectly maintains the hierarchical information of concepts in the original XML document. It is noticed that the related XML data have the identical substring of varied lengths in their SXSs. This identical substring is the Xpath from the root to the NCAN of these related concepts. The ordinal number of the NCAN of the concepts can be used to identify data uniquely. This observation can help retrieve the value of related concepts easily. The system finds data value of related concepts in two substeps: (1) The format of SXS of the non-queried concepts will be decided by using the Xpath associated with the databases of the concepts; (2) Search in the relevant databases for the data whose SXS meets the format specified in the first step.

Consider, for example, if we want to find the name of employees whose has B.S degree. The SXSs of the data valued “B.S” are “R-1-1-1” and “R-3-1-1” in Major.db. The NCAN of Name and Degree are Employee, so the SXS of name value of the employee who has B.S degree should be in the format of “R-1-*” and “R-3-*” in the

Name.db, where * denotes the substring after the employee node in the complete SXS. We then search in the Name.db for the name value whose SXS matches the above two formats, which are “James Wang” and “Linda Lee”.

4.2 Generate Generalized Meta-Patterns

After the raw XML data that are relevant to the AR mining task have been retrieved, they will be usually generalized in order to generate ARs that are significant enough. The generalization of XML data is necessitated by the sparsity of the XML data involved in the AR mining task

It is worthwhile mentioning that the data should be generalized properly in order to find significant yet nontrivial ARs. On one hand, under-generalization may not render the data dense enough for finding patterns which extract significant ARs that can meet support or confidence level. On the other hand, over-generalization may lead to patterns which extract trivial ARs that are not depended on the minimum support and confidence.

Definition 3. ***Primitive meta-patterns*****:** A *primitive meta-pattern* for single-concept AR mining is a tuple *p: {X, a}*, where *X* is the XML data source (IX-tree or Multi-DB) and *a* is the concept involved in the AR mining. A *primitive meta-pattern* for multi-concept AR mining is a tuple *p: {X,* a_1*,...,*a_i*,* s_1*, ...,*s_j*}*, where *X* is the XML data source (IX-tree or Multi-DB), a_1*,...,* a_i are the concepts for the antecedent of the AR, and s_1*,...,*s_j are the concepts for the subsequent of the AR.

Definition 4. ***Generalized primitive meta-patterns*****:** Single-concept generalization involves the generalization of only a concept while in multi-concept generalization more than one concept will be generalized. A *generalized primitive meta-pattern* for single-concept generalization is a tuple *p': {X, a→a'}*, meaning that concept *a* is generalized to concept *a'*. A *k-generalized primitive meta-pattern* for multi-concept generalization ($1 \leqslant k \leqslant n$), denoted as p_k*'*, is a pattern in which *k* concepts are generalized. *n* is the number of concepts in this meta-pattern.

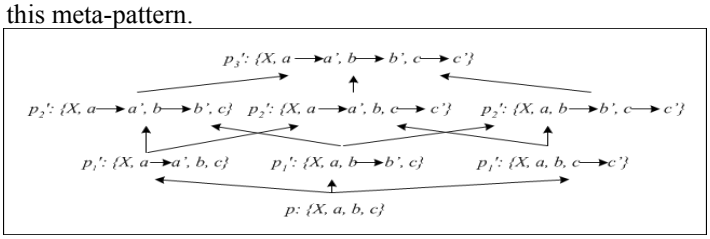

Fig. 4. Generalization lattice of a primitive meta-pattern

Note that there are a combinatorial number of *k*-generalized primitive meta-patterns for a primitive meta-pattern. To depict all the generalized primitive meta-patterns, we use a structure of generalization lattice. Consider, for example, a primitive meta-pattern in the form of *p: {X, a, b, c}*. The corresponding generalization lattice of *p* is shown in Figure 4.

Now, we will present the quantitative generalization metric for measuring the degree of concept generalization. We denote the generalization degree of a generalized meta-patterns *p'* as *Gen(p')* and is defined as follows:

$$Gen(p') = \frac{1}{k} \prod_{i=1}^{k} \overline{D_i} \,/\, D_i$$

where $\overline{D}_i$ denotes the average number of distinct values in each record of the i^{th} concept, D_i denotes the total number of distinct values of the i^{th} concept, and k denote the number of concepts that have been generalized. When k=1, we will obtain the *Gen* value of single-concept generalization.

Now, we will propose the algorithm of finding all the generalized meta-patterns whose generalization degrees fall into the range of [*Min_gen*, *Max_gen*] in multi-concept generalization (the AR mining of single-concept generalization is relatively quite simple since there will be only one generalized meta-pattern). It is known that if p_2 is a derived generalized meta-pattern of p_1, then $Gen(p_1) \leqslant Gen(p_2)$. This theorem can be applied to facilitate the pruning of the excessively generalized meta-patterns: if the generalization degree of a meta-pattern exceeds *Max_gen*, then all its derived generalized meta-patterns are definitely excessively generalized and are therefore pruned away.

Algorithm Find_Generalized_Meta_Patterns (*p, k*)

p—a primitive meta-pattern, k—the number of concepts in p.

Step 1: Initialize a full lattice of p. Compute the generalization degree *Gen* of the primitive meta-pattern p.
IF $Gen(p) \leqslant Min\_gen$, THEN p is pruned from the lattice;
ELSE IF $Gen(p) \geqslant Max\_gen$, THEN return an empty set of generalized meta-patterns and whole algorithm is terminated.

Step 2: FOR i= 1 to k DO
Compute generalization degree *Gen* of i-generalized meta-patterns p_i' of p that has not been pruned from the lattice.
IF $Gen(p_i') \leqslant Min\_gen$, THEN p_i' is pruned from the lattice;
ELSE IF $Gen(p_i') \geqslant Max\_gen$, THEN p_i' and all its derived generalized meta-patterns in the lattice are pruned;

Step 3: Output all the meta-patterns that are not pruned in the lattice.

4.3 Generate Large AR Rules

After the generalized meta-patterns have been obtained, XAR-Miner will generalize the raw XML data based on the meta-patterns and generate large ARs w.r.t. the user-specified minimum support (*minsup*) and confidence (*minconf*) requirements using Apriori algorithm.

Theorem 1: Consider two ARs R_1 and R_2. If R_2 is the generalized AR of R_1, then $\text{Support}(R_1) \leqslant \text{Support}(R_2)$. ■

Based on the above theorem, we can infer that if R_2 is not a large AR, then it is definite that R_1 is not a large AR either. This observation helps us devise an efficient algorithm to perform AR generation of generalized meta-patterns, which adopts a top-down strategy to traverse the generalization meta-patterns. The basic idea is that, instead of directly working on the raw generalized data to generate large ARs for a certain meta-pattern, we can first draw on the large ARs of its higher generalized meta-pattern in order to obtain the candidate ARs and then verify the largeness of these candidate ARs. Obviously, verifying the largeness of ARs is much cheaper than mining all the large AR directly from the data.

Algorithm Find_Large_ARs (*M, n*)

M—the set of generalized meta-patterns obtained, n—the number of meta-patterns in M.

Step1: Sort M in descending order based on the number of generalized concepts of the meta-patterns;

Step 2: FOR i =1 to n do

IF there exists the derived meta-pattern of the i^{th} meta-pattern in the sorted M, THEN candidate ARs of this i^{th} meta-pattern are generated based on the large ARs of its derived meta-pattern and the largeness of candidate ARs is verified;
ELSE mine large ARs of this i^{th} meta-pattern using Apriori algorithm;

Step 3: Output the large ARs of all meta-patterns.

5 Experimental Results

In order to evaluate the performance of XAR-Miner, a comparative study is conducted between XAR-Miner and the method using the MINE RULE operator for AR mining purposes in native XML documents [3] (called MineRule Operator Method), which has to scan the whole XML document in order to select the qualified data for AR and incremental AR mining. The IBM XML Generator [6] is used to generate synthetic XML documents with varying sizes specified by user in our experiments.

To study the efficiency of XAR-Miner in AR mining, we first investigate the execution time of performing an AR mining task from XML documents with varied sizes. From the observation of the experimental results as shown in Figure 5, we can see that the execution time of XAR-Miner using IX-tree and Multi-DB is much smaller than that of the MineRule Operator method. The main reason for this is that each AR mining task in the MineRule Operator method entails a full scan of the XML document while the AR mining in XAR-Miner only requires retrieval of relevant data directly from the IX-tree or Multi-DB, which is obviously more efficient.

Because efficient AR mining in XAR-Miner benefits from the construction of the IX-tree or Multi-DB, the comparison will be more fair if we take into account the cost in this pre-processing step and investigate the total execution time of performing 1-10 and 10-100 AR mining tasks. The total time for performing n AR mining is $T_{average}*n+ T_{pre\text{-}processing}$, where $T_{average}$ denotes the average time spent in each of the AR mining and $T_{pre\text{-}processing}$ denotes the time spent in the pre-processing step, i.e. construction of IX-tree or Multi-DB.

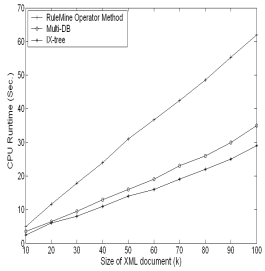

Fig. 5. Mining large ARs from XML document with varied sizes

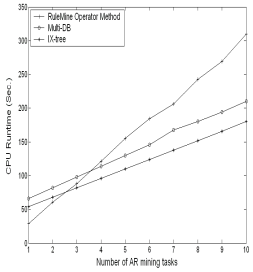

Fig. 6. Performing small number AR mining tasks (1-10)

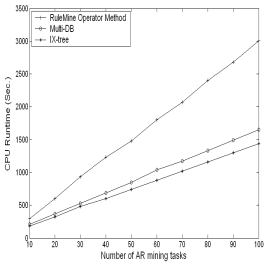

Fig. 7. Performing large number of AR mining tasks (10-100)

XAR-Miner using IX-tree or Multi-DB is larger than that of MineRule Operator method when the number of AR mining tasks to be performed is small (1-3). But as the number of AR mining tasks continues to increase, MineRule Operator begins to require more time than XAR-Miner. This is because that the speedup of XAR-Miner by using IX-tree and Multi-DB overwhelms the cost of constructing IX-tree and Multi-DB when

the number of AR mining tasks becomes large. Figure 7 shows that result comparison when performing larger number of AR mining tasks (10-100), in which the efficiency of XAR-Miner is much better than MineRule Operator method.

6 Conclusions

In this paper, we propose a framework, called XAR-Miner, to mine ARs from XML documents efficiently and effectively. XAR-Miner transform data in the XML document and constructs an Indexed XML Tree (IX-tree) if the XML data can be fully loaded into main memory or Multi-relational databases (Multi-DB) otherwise. The IX-tree and Multi-DB perfectly maintain the hierarchical information of XML data and perform indexing of the data to realize efficient retrieval of data in AR mining. Concepts that are relevant to the AR mining task are generalized to produce generalized meta-patterns. A suitable quantitative metric is devised for measuring the degree of concept generalization in order to prevent under-generalization or over-generalization. An efficient AR mining algorithm is also presented based on candidate AR generation.

References

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases, in Proceedings of *VLDB'94*, pp. 487-499, Santiago de Chile, Chile, Sept. 1994.

[2] A. Amir, R. Feldman and R. Kashi. A New and Versatile Method for Association Generation, in *Information Systems*, Vol. 22(6/7), pp. 333-347, 1997.

[3] D. Braga, A. Campi, M. Klemettinen and P. Lanzi. Mining Association Rules from XML Data, in Proceedings of *DaWaK'02*, Springer LNCS 2454, pp. 21-30, Aix-en-Provence, France, Sept. 2002.

[4] R. Feldman and H. Hirsh. Mining Associations in the Presence of Background Knowledge, in Proceedings of *the 2nd International Conference on Knowledge Discovery in Databases*, pp. 343-346, Portland, Oregon, USA, 1996.

[5] J. Han and M. Kamber. *Data Mining Concepts and Techniques*, Morgan Kaufmann, 2000.

[6] IBM XML Generator. http://www.alphaworks.ibm.com/tech/xmlgenerator.

[7] T. Imielinski and A. Virmani. MSQL: A Query Language for Database Mining, *Data Mining and Knowledge Discovery*, Vol. 3(4), pp. 373-408, Dec. 1999.

[8] R. Meo, G. Psaila and S. Ceri. A New Operator for Mining Association Rules, in Proceeding of *VLDB'96*, pp. 122-133, Bombay, India, Sept. 1996.

[9] R. Meo, G. Psaila and S. Ceri. A Tightly-coupled Architecture for Data Mining, in Proceedings of *ICDE'98*, pp. 316-323, Orlando, FL, USA, Feb. 1998.

[10] PMML 2.0: Predicative Model Makeup Language. Available at http://www.dmg.org, 2000.

[11] P. Resnik. Semantic Similarity in a Taxonomy: An Information-based Measure as its Application to Problems of Ambiguity in Natural Language, in *Journal of Artificial Intelligence Research*, Vol. 11, pp. 95-130, 1999.

[12] L. Singh, B. Chen, R. Haight, and P. Scheuermann. An Algorithm for Constrained Association Rule Mining in Semi-structured Data, in Proceedings of *PAKDD'99*, Springer LNCS 1574, pp. 148-158, Beijing, China, April 1999.

[13] L. Singh, P. Scheuermann and B. Chen. Generating Association Rules from Semi-structured Documents Using an Extended Concept Hierarchy, in Proceedings of *CIKM'97*, pp. 193-200, Las Vegas, Nevada, Nov. 1997.

[14] G. Psaila, and P. L. Lanzi. Hierarchy-based Mining of Association Rules in Data Warehouses, in Proceedings of *ACM SAC'00*, Como, Italy, 2000.

[15] L. Feng, T. S. Dillon, H. Weigand, E. Chang. An XML-Enabled Association Rule Framework. In Proceedings of *DEXA'03*, pp 88-97, Prague, Czech Republic, 2003.

[16] W. W. Wan, G. Dobbie. Extracting association rules from XML documents using XQuery. In Proceedings of *WIDM'03*, pp94-97, New Orleans, Louisiana, USA, 2003.